A SEMANTICS FOR A CLASS OF STRATIFIED PRODUCTION SYSTEM PROGRAMS*

LOUIQA RASCHID

▷ We present our research on defining a correct semantics for forward chaining production system (PS) programs. A correct semantics must ensure that the execution of the program is deterministic and that it will terminate. We define a class of function-free *stratified* PS programs, and develop a fixpoint semantics and a declarative semantics for these programs. A stratified PS program comprises an initial extensional database (EDB_{init}) of facts and a set of productions. We define the conditions for the productions in the PS program to be stratified and we define an operator T_{PS}, which computes the fixpoint for the productions of the stratified PS program. The fixpoint is represented by an updated database EDB_n. A corresponding logic program $\overline{PS}$ is derived from the stratified PS program. $\overline{PS}$ is stratified and has a standard minimal model $M_{\overline{PS}}$. The declarative semantics for the PS program is given by this model $M_{\overline{PS}}$. We prove that the declarative semantics given by the model $M_{\overline{PS}}$ for $\overline{PS}$ is equivalent to the fixpoint EDB_n for productions in the PS program. ◁

1. INTRODUCTION

In recent years, much AI research and development has focused on forward chaining rule-based systems which follow the *production system* (PS) paradigm [10]. Large production rule-based expert systems have been successfully developed in diverse domains such as engineering design databases, troubleshooting in telephone networks, and configuring computer systems [4, 9, 17]. In these domains, the expert

* Preliminary results from this paper were initially published in the Proceedings of the 1990 AAAI Conference [23]. This research is partially sponsored by the National Science Foundation under grant IRI-9008208 and by the Institute of Advanced Computer Studies.

Address correspondence to Louiqa Raschid, Department of Information Systems and Institute for Advanced Computer Studies, University of Maryland, 3129 A. V. Williams Hall, College Park, MD 20742. Email: louiqa@umiacs.umd.edu.

Received October 1990; accepted August 1992.

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/94/$7.00

system programs often have to reason with large quantities of data. As the production rule base and the database grow larger, these programs have to access information stored on disk. Thus, for performance reasons, it is important that PS programs be implemented using database technology. Research in this area is reported in [6, 16, 27, 29, 32].

Fortunately, the forward chaining PS paradigm has some similarities with processing in a database management system (DBMS). For example, triggers and integrity constraints are very important in DBMS research; they are activated in response to updates made to the database. Similarly, in the PS paradigm, the selection and execution of production rules are activated as a result of making updates to the database. In a DBMS, the execution of update operations in a transaction changes the database. Similarly, execution of production rules in a PS program can result in updates to the database as well. Notwithstanding such similarities, the execution semantics of production rules in a PS program are more complex than the execution of a single DBMS transaction. In the context of a PS program, production rules are repeatedly matched against the database and selected production rules are executed, as a result of which the database is updated. This in turn, causes other production rules to be selected for execution. This process continues until no further production rules can be executed and a fixpoint is reached. Thus, if PS programs are to be implemented to interface correctly with large (relational) databases, then it is critical that the semantics of executing a sequence of production rules in a program, against the database, be well understood.

Unfortunately, most PS programs have an incomplete operational semantics defined for them. When the execution of production rules in the PS program updates the database, there is no semantics defined for checking the correctness of such updates. This can often result in nonterminating execution of production rules and may produce different answers from the same PS program.

The success in implementing systems that integrate first order Horn logic programs with function-free first order relational databases [2, 3, 11–13, 18, 19, 30] can be largely attributed to the fact that first order logic is the theoretical foundation for both systems. Horn logic programs have defined for them a fixpoint semantics, a model-theoretic semantics, and an operational semantics, which are all equivalent. Relational databases correspondingly have a proof-theoretic and a model-theoretic semantics [31]. However, when logic programs are interfaced to relational databases, the semantics for updating the database through the rules of the logic program is not always well defined. Since updating the database through executing the production rules of a PS program is the basis for defining the operational behavior of PS programs, it is clear that the semantics for updating the database is an important issue and must be investigated.

In this paper, we describe our research on defining a correct semantics for executing PS programs in a DBMS. By a correct semantics, we mean that the execution of a PS program must be repeatable and must not produce different answers, i.e., it must be deterministic. Further, the execution of the program must terminate. We define such a class of programs called stratified PS programs. A stratified PS program is a function-free program, and comprises an initial extensional database (EDB_{init}) of facts and a set of production rules. We define the conditions for the production rules in the PS program to be stratified. We show that processing is guaranteed to terminate upon reaching the fixpoint of a defined

operator T_{PS}. The fixpoint, represented by the updated extensional database EDB_n, captures the answers that are produced by the stratified PS program.

In computing a fixpoint, the production rules update the relations in the database, and the objective of this research is to provide a correct semantics for these updates. We make use of stratification together with the definition of the fixpoint operator to ensure repeatable terminating behavior of the PS programs. However, there may be several other approaches for ensuring these properties, each associated with a different semantics for executing the production rules. For this reason, we provide a declarative or minimal model semantics which is equivalent to and explains the behavior of the execution of the production rules.

To obtain an equivalent declarative semantics, a stratified logic program $\overline{PS}$ is associated with each PS program. The production rules are used to derive a set of Horn rules of logic program and a set of integrity constraints. We explain the semantics of making updates to the database in terms of maintaining consistency with the constraints derived from the production rules. The corresponding stratified logic program $\overline{PS}$ has a standard minimal model $M_{\overline{PS}}$ [1]. The declarative semantics for the stratified production rule program PS is given by this minimal model.

The advantage of associating a logic program with a stratified PS program is this definition of a declarative semantics. For example, the use of a declarative semantics provides an alternative method of determining the equivalence of two PS programs if they produce the same minimal model. This can be contrasted with other methods which have been proposed for determining the equivalence of production rule programs. In addition, by associating the declarative semantics of the equivalent logic program $\overline{PS}$ with each production rule program PS, we can obtain a better understanding of the PS program, especially when the behavior of the PS program in not deterministic. This has been explored in research on nondeterministic and causal PS programs, discussed in [25], and in identifying PS programs that have stable models, discussed in [26]. In each of these cases, the declarative semantics of an equivalent logic program $\overline{PS}$ was used to identify classes of acceptable production rule programs and to explain the behavior of each of these classes of PS programs.

We define a transformation to obtain a stratified logic program, $\overline{PS}$, from the stratified PS program. Since $\overline{PS}$ represents the PS program, the same answers must be obtained in the fixpoint semantics for the PS program or from the declarative semantics for $\overline{PS}$. Thus, for the class of function-free stratified PS programs, we must show that the standard minimal model $M_{\overline{PS}}$ for $\overline{PS}$ is equivalent to EDB_n, the fixpoint for the productions in the stratified PS program.

This paper is organized as follows: In Section 2, we introduce the syntax for defining production rules. We identify some example programs which are not deterministic or result in nontermination of program execution. Section 3 presents results on the fixpoint semantics for stratified PS programs. An example PS program is used to demonstrate obtaining a stratification and computing the fixpoint EDB_n. In Section 4, we describe the syntactic transformation to derive a stratified logic program $\overline{PS}$ from a PS program. We use the example PS program to demonstrate this syntactic transformation. In Section 5, we demonstrate the equivalence between the fixpoint semantics for the stratified PS program and the declarative semantics for the corresponding logic program $\overline{PS}$. We first show that when PS is stratified, then $\overline{PS}$ is also stratified. We then prove that the updated

extensional database EDB_n, i.e., the fixpoint for the productions in PS, and the standard model $M_{\overline{PS}}$ for the stratified logic program $\overline{PS}$, are equivalent. Section 6 is a summary and discusses future research.

2. SYNTAX FOR PRODUCTION RULE PROGRAMS AND EXAMPLE PROGRAMS

In this section we introduce the syntax for specifying production rules of a PS program. We use a syntax that is similar to the OPS5 production system language [7, 8].

2.1. Syntax for PS Programs

A *production rule* consists of (1) the *name*, (2) the antecedent on the left-hand side (LHS) (also referred to as the *body* of the production rule), (3) the symbol $\rightarrow$, and (4) the consequent actions on the right-hand side (RHS) (also known as the *head* of the production rule).

The body is a conjunction of first order positive literals of the form $P(\bar{u})$ or negative literals of the form $\neg Q(\bar{v})$. The actions in the head are of the form **assert** $R(\bar{u})$ or **retract** $S(\bar{v})$. P, Q, R, and S are predicates corresponding to database relations of the same arity, and $\bar{u}$ and $\bar{v}$ are vectors of terms from a nonempty finite or infinite set of constants C or a set of variable V.

We assume that all variables are *range-restricted* [5, 20, 21], i.e., any variable that occurs in a literal must appear in a positive literal in the body of a production rule. The advantages of this restriction correspond to the safety of evaluating queries. It also ensures that only ground atoms are inserted into the databse. The syntax restricts the atom referred to by the **retract** action, to occur in the body of that production rule, so that only ground atoms that are in the database are retracted. However, this restriction is syntactic and may be relaxed as needed.

In general, the body of production rules may include *functions* in the form of *evaluable* predicates of range-restricted variables. The head may also include a sequence of actions. We have not considered the effect of such extensions in this paper; however, sequences of actions in the head can easily be accommodated by a straightforward rewriting of the rules. These issues are discussed in related research [24].

A function-free *production rule program PS* consists of the following:

1. A set of *a-productions*, each of which has a *single* **assert** action in its head.
2. A set of *r-productions*, each of which has a *single* **retract** action in its head.
3. An initial extensional database of ground atoms EDB_{init}.

Processing in a PS program can be informally described as follows: The body of each production rule is interpreted as a query against the database relations. For example, for each of the positive literals, $P(\bar{u})$, relation P is queried and a set of *instantiated* tuples of P satisfying each positive literal in the body is retrieved. For each of the negative literals, $\neg Q(\bar{v})$, the query is verified against the corresponding relation Q. Each variable x in $\bar{v}$ is range restricted to the value obtained by evaluating a query for some positive literal P in which x occurs. The body of a production rule is *satisfied* if the relations contain instantiated tuples correspond-

ing to each of the positive literals and if the relations do not contain tuples corresponding to the negative literals.

Production rules are selected for execution and they update the database. Depending on the action in the head of the selected production rule, the database relations are updated. Either new facts are asserted by the a-productions, i.e., the tuples are inserted into the corresponding relation, or existing facts are retracted by the r-productions, i.e., the tuples are deleted from the relation. This process continues until the relations can no longer be updated.

2.2. *Problems in Executing Example Production Rule Programs*

The first problem is that the execution of a program may not terminate and the relations may be updated indefinitely. The second problem is that an initial database of relations and a PS program can sometimes produce different answers, i.e., its behavior may be nondeterministic. We present some motivating example programs that highlight these problems.

Example 1. Consider a PS program whose initial database has two tuples, {Employee(Mike). GoodWorker(Mike).}, and the following set of production rules:

(p_1 Employee(X), GoodWorker(X) → **assert** Manager(X))
(p_2 Manager(X) → **assert** HasOffice(X))
(p_3 Employee(X), HasOffice(X) → **assert** PoorWorker(X))
(p_4 Manager(X), PoorWorker(X) → **retract** Manager(X))

Given this initial database and corresponding set of production rules, the production rules will execute until a fixpoint is reached. Production rules p_1, p_2, and p_3 will execute in that sequence and the tuples Manager(Mike), HasOffice(Mike), and PoorWorker(Mike) will be added to the corresponding EDB relations, Manager, HasOffice, and PoorWorker. Next, production rule p_4 executes and the tuple Manager(Mike) will be deleted from the manager relation. Subsequently, p_1 and p_4 will execute, first inserting the tuple Manager(Mike) and then deleting this tuple from the Manager relation. Processing of these production rules p_1 and p_4 will continue indefinitely.

Example 2. The initial database is {Employee(Mike).} and the production rules are as follows:

(p_1 Employee(X) → **assert** GoodWorker(X))
(p_2 Employee(X), GoodWorker(X) → **assert** Manager(X))
(p_3 Employee(X), ¬ GoodWorker(X) → **assert** PoorWorker(X))

If production rules are executed in the sequence p_1 followed by p_2, then the final database will contain the set of tuples {Employee(Mike). GoodWorker(Mike). Manager(Mike).}. If the execution sequence was p_3 followed by p_1 and p_2, then the final database would include the tuples {Employee(Mike). GoodWorker(Mike). Manager(Mike). PoorWorker(Mike).}. In this case, the program has two different fixpoints. The first is a subset of the second and so the second fixpoint is not

minimal. PoorWorker(Mike) is true in one fixpoint, but is false in the other, and this could lead to an inconsistency.

3. A FIXPOINT SEMANTICS FOR STRATIFIED PRODUCTION RULE PROGRAMS

We define a class of function-free stratified PS programs and develop a fixpoint semantics for these programs. The fixpoint semantics is based on an operator T_{PS} which is used to compute a fixpoint for the production rules of the program. Processing is guaranteed to terminate upon reaching this fixpoint.

Stratified logic programs are an extension of Horn programs to more general Horn programs to allow negative literals in the body of a rule. Since we apply the concept of stratification of logic programs to PS programs, we informally define it in this section.

A general Horn logic program P consists of a finite set of rules of the following form:

$$A \leftarrow L_1, L_2, \ldots, L_m,$$

where A is an atom and each of the L_i are literals. If $m = 0$, then A is a fact. P is stratified if there exists a *stratification* $P = P_1 \dot{\cup} \cdots \dot{\cup} P_n$, where $\dot{\cup}$ is a disjunctive union, such that the following conditions hold for $i = 1, 2, \ldots, n$:

1. If a predicate occurs *positively* in the body of a clause in P_i, then its definition is contained within $\cup_{j \leq i} P_j$. The definition (of a predicate) is all clauses in which the predicate symbol occurs in the head of the clause.
2. If a predicate occurs *negative* in the body of a clause in P_i, then its definition is contained within $\cup_{j < i} P_j$. P_1 can be empty. We say that P is stratified by $P_1 \dot{\cup} \cdots \dot{\cup} P_n$ and each P_i is called a stratum of P. Thus, each stratum defines new relations in terms of itself and *other* relations in the same stratum only positively and in terms of the relations from the previous strata, possibly negatively.

The meaning of a stratified program is given by minimal (supported) models. A model theory for stratified programs is presented in [1], where it is shown that if P is a stratified program, then there exists a standard minimal supported model M_P for P.

3.1. A Stratified PS program

A stratified production system program PS is function-free and comprises the following:

1. A set of a-productions, each of which has a *single* **assert** action in its consequent.
2. A set of r-productions, each of which has a *single* **retract** action in its consequent.
3. An initial database of facts (EDB_{init}).

There must exist a partition so that PS is a stratified program. Thus, $PS = PS_0 \dot{\cup} PS_1 \cdots \dot{\cup} PS_n$. Each of the partitions PS_i comprises a set of a-productions and a set

of r-productions. The sets of a-productions or r-productions in each partition may be empty. Partition PS_0 comprises the initial database and there are no productions rules in PS_0. Let the production rules in PS be as follows:

$(p_a \ A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b \rightarrow \textbf{assert } P)$. or

$(p_r \ P, C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow \textbf{retract } P)$.

Then, the following conditions must hold for the stratification of the PS program:

1. $PS = PS_0 \dot{\cup} PS_1 \dot{\cup} \cdots \dot{\cup} PS_n$.
2. For every predicate A_k (or C_k) occurring positively in the body of a production rule such as p_a or p_r, in PS_i, all a-productions in which A_k (or C_k) occurs in the **assert** action, must be contained within $\bigcup_{j \leq i} PS_j$.
3. For every predicate A_k (or C_k) occurring positively in the body of a production rule in PS_i, all r-productions in which A_k (or C_k) occurs in the **retract** action must be contained within $\bigcup_{j < i} PS_j$.
4. For every predicate B_k (or D_k) occurring negatively in the body of a production rule in PS_i, all a-productions in which B_k (or D_k) occurs in the **assert** action must be contained within $\bigcup_{j < i} PS_j$.
5. For every predicate B_k (or D_k) occurring negatively in the body of a production rule in PS_i, all r-productions where B_k (or D_k) occurs in the **retract** action must be contained within $\bigcup_{j < i} PS_j$.
6. Finally, for the predicate P associated with the **retract** action of an r-production such as p_r in PS_i, all production rules with P occurring in the head must be contained within $\bigcup_{j \leq i} PS_j$.

Note that condition 6 allows us to relax the restriction that P must occur in the body of an r-production such as p_r that has (**retract** P) in its head. This condition will place all r-productions with (**retract** P) in their heads in the same partition.

3.2. *The Operator T_{PS}*

Once such a partition has been obtained for the production rules in the stratified PS program, then we define an operator T_{PS} which computes a fixpoint for the production rules in each partition PS_i of PS.

Consider the domain, D_{PS} for a PS program to be the set of all possible ground atoms of the form **assert** $P(\bar{t})$ and **retract** $P(\bar{t})$, where P is any predicate in the PS program and $\bar{t}$ is a vector of constants. We use a set of relations to represent D_{PS}, with two relations corresponding to each predicate symbol. Thus, for predicate P, relation **assertP** will contain all tuples $P(\bar{t})$ which are associated with some action (**assert** $P(\bar{t})$) and relation **retractP** will contain all tuples $P(\bar{t})$ which are associated with some action (**retract** $P(\bar{t})$). If we consider a propositional PS program where the propositional variables are the set $\{A, B, C\}$, then D_{PS} is the set {**(assert A)**, **(retract A)**, **(assert B)**, **(retract B)**, **(assert C)**, **(retract C)**}.

Definition

$$EDB_0 = \{\textbf{assert } P(\bar{t}) \mid P(\bar{t}) \in EDB_{init}\}.$$

Thus, for each ground atom $P(\bar{t})$ in EDB_{init}, we add the tuple $P(\bar{t})$ to the relation **assertP** in EDB_0. In the propositional case, for each variable P in EDB_{init}, we add (**assert** P) to EDB_0.

Definition. Given a subset S of D_{PS}, we define the following *entailment* relation:

- S entails $P(\bar{t})$ if **assert** $P(\bar{t}) \in S$ and **retract** $P(\bar{t}) \notin S$.
- S entails $\neg P(\bar{t})$ if and only if S does not entail $P(\bar{t})$. This occurs with the following:

  assert $P(\bar{t}) \in S$ and **retract** $P(\bar{t}) \in S$ or

  assert $P(\bar{t}) \notin S$ and **retract** $P(\bar{t}) \notin S$.

- The following cannot occur in S: **assert** $P(\bar{t}) \notin S$ and **retract** $P(\bar{t}) \in S$. This is due to the syntactic restriction on the language.

We define an operator T_{PS} which, when applied to the production rules in a partition PS_i of the PS program, will update the relations for each EDB_i. T_{PS} is defined as follows:

Definition. Let $T_{\text{PS}}: 2^{D_{\text{PS}}} \rightarrow 2^{D_{\text{PS}}}$ be a mapping from a subset of D_{PS} to a subset of D_{PS}, defined as follows, where S is a subset of D_{PS}:

$$
\begin{aligned}
T_{\text{PS}}(S) = \{&\textbf{assert } P(\bar{t}) \mid \text{there is a ground instance} \\
&p\colon B_1, B_2, \ldots, B_b, \neg C_1, \neg C_2, \ldots, \neg C_c \rightarrow \textbf{assert } P(\bar{t}) \\
&\text{of a production rule in PS such that } S \text{ entails} \\
&B_1, B_2, \ldots, B_b, \neg C_1, \neg C_2, \ldots, \neg C_c\} \\
\cup \{&\textbf{retract } P(\bar{t}) \mid \text{there is a ground instance} \\
&p\colon B_1, B_2, \ldots, B_b, \neg C_1, \neg C_2, \ldots, \neg C_c \rightarrow \textbf{retract } P(\bar{t}) \\
&\text{of a production rule in PS such that } S \text{ entails} \\
&B_1, B_2, \ldots, B_b, \neg C_1, \neg C_2, \ldots, \neg C_c\} \\
\cup\, &S
\end{aligned}
$$

The iterative application of the operator for the production rules in a partition is

$$T_{\text{PS}} \uparrow 0(S) = S,$$

$$T_{\text{PS}} \uparrow n(S) = T_{\text{PS}} \uparrow 1\,(T_{\text{PS}} \uparrow n - 1(S)), \qquad n > 0.$$

3.3. Obtaining a Fixpoint for T_{PS}

Processing for each partition PS_i should terminate when a fixpoint is reached, i.e., when there are no longer any production rules in PS_i that can update the relations in EDB_i. Processing for PS terminates when the fixpoint for production rules in PS_n is reached and EDB_n is computed.

Theorem. $\mathrm{EDB}_i = T_{\mathrm{PS}_i} \uparrow \omega\ (\mathrm{EDB}_{i-1})$ *is a fixpoint of* T_{PS_i}, *for* $i = 1, 2, \ldots, n$. *We observe from the definition of the operator* T_{PS} *that it is growing, i.e.,* $S \subseteq T_{\mathrm{PS}}(S)$ *and* $T_{\mathrm{PS}} \uparrow n(S) \subseteq T_{\mathrm{PS}} \uparrow 1\ (T_{\mathrm{PS}} \uparrow n(S))$. *Since the domain* D_{PS} *for the function-free PS program is finite, after a finite number of applications,* $< \omega$, *applying the operator will not add any additional elements of the form* (**assert** $P(\bar{t})$) *or* (**retract** $P(\bar{t})$) *to* EDB_i.

Thus, $\mathrm{EDB}_i = T_{\mathrm{PS}_i} \uparrow \omega(\mathrm{EDB}_{i-1})$ *is a fixpoint of* T_{PS_i} *for* $1 = 1, 2, \ldots, n$.

EDB_i is computed as follows, iteratively applying the operator T_{PS} *to the production rules in each partition* PS_i *until a fixpoint is reached for that partition, after applying the operator a finite number of times*:

$$\mathrm{EDB}_1 = T_{\mathrm{PS}_1} \uparrow \omega(\mathrm{EDB}_0),$$

$$\mathrm{EDB}_2 = T_{\mathrm{PS}_2} \uparrow \omega(\mathrm{EDB}_1),$$

$$\vdots$$

$$\mathrm{EDB}_n = T_{\mathrm{PS}_n} \uparrow \omega(\mathrm{EDB}_{n-1}).$$

3.4. Computing a Fixpoint for an Example Stratified PS program

We use an example PS program to demonstrate how the program is stratified and the process of computing a fixpint for the production rules.

Example 3. Consider the set of production rules in Table 1. The centered column represents the conditions that must be satisfied to partition the production rules and the number on the right is the corresponding stratum for the production rule, obtained after solving the inequalities of the center column and partitioning the PS program (note that there may be more than one partitioning).

The stratified PS program is partitioned as follows:

$$\mathrm{PS}_1 = \{p_1, p_2, p_3, p_5, p_7\},$$

$$\mathrm{PS}_2 = \{p_4, p_6, p_8\},$$

$$\mathrm{PS}_3 = \{p_9\}.$$

If we consider an initial database $\mathrm{EDB}_{\mathrm{init}}$ containing $\{P, A, B, F\}$, then EDB_0 is as follows:

$$\mathrm{EDB}_0 = \{(\textbf{assert } P).\ (\textbf{assert } A).\ (\textbf{assert } B).\ (\textbf{assert } F).\},$$

TABLE 1.

Production rule	Constraint for partition	Stratum
(p_1 $A, B \rightarrow$ **assert** C)	i	1
(p_2 $C \rightarrow$ **assert** D)	$j \geq i$;	1
(p_3 $C \rightarrow$ **assert** E)	$k \geq i$;	1
(p_4 $\neg D \rightarrow$ **assert** F)	$l > j$;	2
(p_5 $D \rightarrow$ **assert** G)	$m \geq j$;	1
(p_6 $\neg E \rightarrow$ **assert** G)	$n > k$;	2
(p_7 $E \rightarrow$ **assert** F)	$o \geq k$;	1
(p_8 $P, F \rightarrow$ **retract** F)	$p \geq l$; $p \geq o$;	2
(p_9 $F, G \rightarrow$ **retract** G)	$q \geq l$; $q \geq m$; $q \geq n$; $q \geq o$; $q > p$;	3

The fixpoint for the production rules in PS_1, EDB_1, is as follows:

$$EDB_1 = \{(\textbf{assert } P).(\textbf{assert } A).(\textbf{assert } B).(\textbf{assert } F).$$
$$(\textbf{assert } C).(\textbf{assert } D).(\textbf{assert } E).(\textbf{assert } G).\}.$$

Next, when computing the fixpiont for the production rules in PS_2, the production rule p_8 will add (**retract** F) to EDB_2. EDB_2 is as follows:

$$\{(\textbf{assert } P).(\textbf{assert } A).(\textbf{assert } B).(\textbf{assert } F).(\textbf{assert } C).$$
$$(\textbf{assert } D).(\textbf{assert } E).(\textbf{assert } G).(\textbf{retract } F).\}.$$

EDB_3 is identical to EDB_2, since the production rule p_9 in PS_3 is not executed. Thus, the fixpiont EDB_3 for the PS program will entail

$$\{P.\ A.\ B.\ C.\ D.\ E.\ G.\}.$$

To summarize, we have defined the conditions to obtain a stratified PS program as well as a fixpoint semantics for the PS program based on a fixpoint operator T_{PS}. Applying the operator to the production rules in each partition, iteratively, will compute a fixpiont EDB_n for the PS program.

4. OBTAINING A CORRESPONDING LOGIC PROGRAM

To provide a declarative semantics for stratified PS programs, we associate a stratified logic program $\overline{PS}$ with the PS program. $\overline{PS}$ is derived from the production rules of the PS program and from the initial database EDB_{init}. The *a*-productions in the PS program seem to naturally derive general Horn rules. The *r*-productions are syntactically similar to *denial* integrity constraints proposed in research by Kowalski and Sadri [14, 15].

A *denial integrity constraint* is given by a clause, representing the *denial*, followed by a *retractable* atom which is enclosed within [ ], as follows:

$$L_1, L_2, \ldots, L_n \rightarrow .\,[L_i].$$

The meaning associated with this constraint is that $L_1 \wedge L_2 \wedge \cdots \wedge L_n$ cannot be simultaneously true in the database if the database is to be consistent with the denial. If the denial is violated in the database, then consistency can be restored by making one of the positive literals in the denial false. A single atom L_i which occurs in the denial is chosen as the retractable atom, and it is specified as such in the constraint.

If we examine the *r*-production

$$(p_i \quad P, C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow \textbf{retract } P),$$

the operational meaning is that the atom P is retracted from the database when the database entails P and each C_k, but does not entail each D_k. In other words, if we consider the conjunction in the body of the *r*-production to correspond to a denial, then when this denial is violated, the atom P is retracted. Since the atom P which is retracted by the *r*-production must occur in the conjunction in the body of the *r*-production, *r*-productions are syntactically similar to the denial integrity constraints. We note that even if we relax the restriction that the retractable atom

must occur in the body of the r-production, the meaning of the constraint remains unchanged.

There has been considerable research on the problem of maintaining consistency in databases with respect to a set of integrity constraints. Informally, a database must satisfy its integrity constraints as it changes over time. Usually, an update to the database (more precisely an update to facts in the database) may cause the violation of an integrity constraint; such updates must be rejected or modified. Sometimes, the database itself, i.e., the facts and the rules, may be inconsistent with the constraints, and the database must be modified to maintain consistency with the constraints. This is the approach that we have chosen.

In [14, 15] a method is presented for transforming a stratified logic program whose standard model is inconsistent with respect to a set of denial integrity constraints into a transformed program whose model is consistent with the constraints. The transformation is syntactic. We note that the transformation works for constraints that are more general than the denial integrity constraints. We explain the transformation using an example, as follows:

Consider an initial program $\{C.\ C \rightarrow A.\}$ and the denial integrity constraint $\{A, \neg B \rightarrow .[A]\}$. The constraint states that when the database proves A but not B, then A must be retracted. Thus, the standard model for this initial program, $\{C.\ A.\}$ is inconsistent with the constraint. The transformation to maintain the constraint gives the transformed program $\{C.\ C, B \rightarrow A.\}$, where the rule defining A has been changed. This program has a standard model, $\{C.\}$, which is consistent with the constraint.

4.1. *The Transformation to Derive* $\overline{\text{PS}}$

The transformation from a stratified PS program to a logic program $\overline{\text{PS}}$ takes place in two steps. Initially, a stratified logic program PS_{init} is derived from the initial database EDB_{init} and the a-productions of the stratified PS program. The r-productions are used to derive a set of denial integrity constraints IC. This transformation is TR_{init}. However, the standard model for PS_{init} may be inconsistent with the set IC. The second transformation TR_{cons} is similar to the transformation of [14]. It goes from PS_{init} to $\overline{\text{PS}}$, using IC. The final logic program $\overline{\text{PS}}$ then represents the meaning of the stratified PS program. Figure 1 is a graphical representation of the two transformations to obtain $\overline{\text{PS}}$.

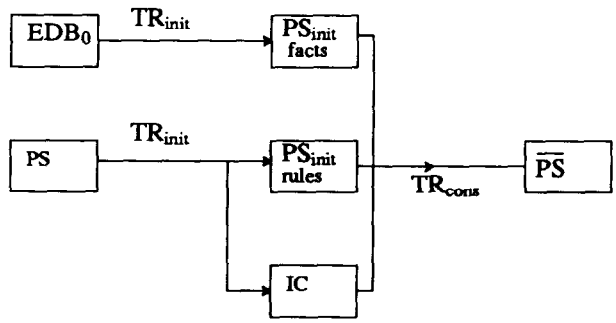

FIGURE 1. The transformations TR_{init} and TR_{cons} to obtain $\overline{\text{PS}}$ from PS.

4.2. *The Transformation* $\mathrm{TR}_{\mathrm{init}}$

The transformation $\mathrm{TR}_{\mathrm{init}}$ is given by the following definitions.

Definitions

Every tuple in the initial EDB relations is a fact of the initial program $\mathrm{PS}_{\mathrm{init}}$.

Every a-production derives a rule of the initial program ($\mathrm{PS}_{\mathrm{init}}$) defining the predicate named in the head of the a-production.

Every r-production derives a denial integrity constraint of the set IC. The literal P associated with the **retract** action in the head of the r-production is the retractable literal.

For example, the r-production

$$(p_r \quad P, C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow \textbf{retract } P)$$

derives the IC

$$P, C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow .\,[P].$$

4.3. *The Transformation Algorithm* $\mathrm{TR}_{\mathrm{cons}}$

This algorithm for the transformation $\mathrm{TR}_{\mathrm{cons}}$ uses as input the logic program $\mathrm{PS}_{\mathrm{init}}$ and the set of denial integrity constraints IC and produces the logic program $\overline{\mathrm{PS}}$ as output. The algorithm proceeds as follows:

Step 1. Each denial integrity constraint q in IC, where $P(\bar{v})$[1] is the retractable literal, transforms each rule p in $\mathrm{PS}_{\mathrm{init}}$ which defines $P(\bar{u})$ or the fact $P(\bar{u})$. Two special unused predicates P^* and P^{**} are associated with each predicate P. P^* corresponds to the case when $\bar{u}$ and $\bar{v}$ are unifiable, and P^{**} corresponds to the case when these two terms are not unifiable.

Let the a-production p be of the form

$$p\colon A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b \rightarrow \textbf{assert } P(\bar{u})$$

where a or b could be equal to 0, i.e., $P(\bar{u})$ could be a fact. Let the corresponding denial IC q be

$$q\colon P(\bar{v}), C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow .\,[P(\bar{v})].$$

Let $\bar{u}$ and $\bar{v}$ unify with the most general unifier θ. Then the following rules are placed in $\overline{\mathrm{PS}}$:

$$C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d, \rightarrow P^*(\bar{v}),$$

$$\neg P^*(\bar{v})\theta, A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b, \rightarrow P(\bar{u})\theta,$$

$$\neg P^{**}(\bar{u}), A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b, \rightarrow P(\bar{u}),$$

$$(\bar{v} = \bar{u}) \rightarrow P^{**}(\bar{u}).$$

[1] $\bar{v}$ represents the vector of terms $(\overline{v_1}, \ldots, \overline{v_n})$.

Step 2. Each fact or rule in PS_{init} which is not modified by a denial integrity constraint in IC is placed unchanged in $\overline{PS}$.

In the rest of this paper, we simplify the discussion and assume that the terms $\bar{u}$ and $\bar{v}$, associated with each pair of a-productions and r-productions, respectively, are always unifiable. Thus, only a single special predicate P^* will be associated with some predicate P in PS, instead of a second special predicate P^{**}, as was used in Step 1 of the transformation TR_{cons}.

4.4. *An Example of Deriving* $\overline{PS}$

We use the previous example to derive $\overline{PS}$ from the stratified PS program.

Example 3 (continued). An initial logic program PS_{init} and a set of denial integrity constraints (IC) are derived from the example PS program. The facts in PS_{init} correspond to the initial database EDB_{init}. The rules in PS_{init}, are derived from the a-productions in PS, and the IC are derived from the r-productions, using TR_{init}, as follows:

$$PS_{init} = \{P.A.B.F.\} \cup \{A, B \rightarrow C.\ C \rightarrow D.\ C \rightarrow E.\ \neg D \rightarrow F.\ D \rightarrow G.\ \neg E \rightarrow G.\ E \rightarrow F.\},$$

$$IC = \{P, F \rightarrow .[F] \cup F, G \rightarrow .[G]\}.$$

The standard model for this program PS_{init} is $\{P.A.B.F.C.D.E.G.\}$ and it is inconsistent with the set of integrity constraints IC. The transformation TR_{cons} is used to obtain $\overline{PS}$. Subsequently, $\overline{PS}$ is stratified as defined in [1]: $\overline{PS} = \overline{PS_0} \cup \overline{PS_1} \cup \overline{PS_2} \cup \overline{PS_3}$. The standard model for $\overline{PS}$ is consistent with IC. Since this example is in the propositional case, there is no need to use the special predicates during the transformation (we note that there could be more than one stratification possible):

$$\overline{PS_0} = \{P.A.B.\},$$

$$\overline{PS_1} = \{\neg P \rightarrow F.\ A, B \rightarrow C.\ C \rightarrow D.\ C \rightarrow E.\ E, \neg P \rightarrow F.\},$$

$$\overline{PS_2} = \{\neg D, \neg P \rightarrow F.\},$$

$$\overline{PS_3} = \{\neg E, \neg F \rightarrow G.\ D, \neg F \rightarrow G.\}.$$

The standard model for the stratified program $\overline{PS}$ is

$$\{P.A.B.C.D.E.G.\}.$$

It can be seen, in this example, that the standard model for $\overline{PS}$ is consistent with IC, and the operational fixpoint EDB_n for PS entails the standard model for the logic program $\overline{PS}$.

5. PROVING THE EQUIVALENCE OF THE FIXPOINT AND DECLARATIVE SEMANTICS

We have shown that each stratified PS program can be associated with a stratified logic program $\overline{PS}$. The declarative semantics for the PS program is given by the standard minimal model $M_{\overline{PS}}$ for $\overline{PS}$. The transformation is such that $M_{\overline{PS}}$ is also

consistent with the integrity constraints IC. This is a result from [14]. For the fixpoint semantics and the declarative semantics to be equivalent, we must show that the fixpoint EDB_n for the PS program entails the minimal model $M_{\overline{PS}}$ for $\overline{PS}$, ignoring the special predicates in $\overline{PS}$ which do not occur in PS. These results are formally proved in this section. Figure 2 provides a graphical description of this equivalence.

To explain this equivalence, informally, consider the execution of production rules in the PS program. When there exists a production rule in PS_i and when the literals in the body of this production rule are entailed by applying the T_{PS} operator some n ($< \omega$) times to EDB_{i-1}, then the action in the head of this production rule updates the database. If it is an a-production, then some new fact is added to the database. When an r-production in PS_i is executed, then we say that (the entailment of) the database satisfied the literals in the body of this r-production and it violated the denial in the body of the corresponding integrity constraint. The corrective action itself is determined by the **retract** action in the head of the r-production that is executed, and some fact is retracted from the database.

Assume that the head of the r-production in PS_i that derives the violated integrity constraint is (**retract** P). We know that P must occur as a positive literal in the body of this production rule. This means that either (**assert** P) $\in EDB_0$ or some a-production in $\bigcup_{j \leq i} PS_j$ added (**assert** P), so that (**assert** P) $\in EDB_i$. Subsequently, the r-production in PS_i violated a constraint and added (**retract** P) to EDB_i to restore consistency. Now, (**assert** P) $\in EDB_i$ and (**retract** P) $\in EDB_i$. Consequently, EDB_i and EDB_n will not entail P. For the corresponding declarative semantics, the logic program $\overline{PS}$ must not prove P if it is to be consistent with the integrity constraints IC. However, PS_{init} derived from the a-productions of the PS program and the initial database may prove P. If this is the case, then PS_{init} must be modified so that $\overline{PS}$ cannot prove P. This is achieved through the transformation TR_{cons}, described in the previous section.

If the fixpiont semantics and the declarative semantics are equivalent, given $M_{\overline{PS}}$ is a model for $\overline{PS}$, then the fixpoint for PS, EDB_n, must entail $M_{\overline{PS}}$, ignoring the

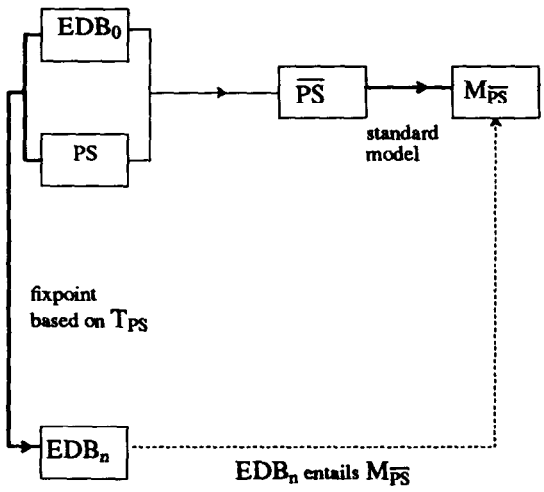

FIGURE 2. Equivalence of the fixpoint semantics for PS and the declarative semantics for $\overline{PS}$.

new predicates introduced in $\overline{\text{PS}}$. To elaborate, when P is true in $M_{\overline{\text{PS}}}$, then EDB_n must entail P, and when P is not true in $M_{\overline{\text{PS}}}$, then EDB_n must not entail P. We will prove in this section that if the PS program is stratified, then the logic program $\overline{\text{PS}}$ is stratified. Next, we will prove that the fixpoint EDB_n entails the model $M_{\overline{\text{PS}}}$ for $\overline{\text{PS}}$, ignoring the special predicates in $\overline{\text{PS}}$.

Definitions

For two predicates P and Q, P *p-derives* Q in a logic program, denoted $P \rightarrow^{+} Q$, if P occurs positively in the body of a rule in the program defining Q or if P occurs positively in the body of a rule defining X, and $X \rightarrow^{+} Q$ in the program.

P *n-derives* Q[3] in a logic program, denoted $p \rightarrow^{-} Q$, if $\neg P$ occurs in the body of a rule in the program defining Q or

if P occurs positively in the body of a rule defining X and $X \rightarrow^{-} Q$ in the program or

if $\neg P$ occurs in the body of a rule defining literal X and either $X \rightarrow^{+} Q$ or $X \rightarrow^{-} Q$.[2]

Definitions. A predicate P is *distinguished* in partition PS_j of a production rule program PS if P occurs in the body of a rule p in PS_j.

A production rule p is *relevant* to predicate P if P occurs in the head of the rule p.

A predicate P is *retractable* in partition PS_j if there is an r-production p in PS_j relevant to P.

Lemma 1. Let $\overline{\text{PS}}$ *be the logic program derived from* PS *and let* $P \rightarrow^{+} Q$ *or* $P \rightarrow^{-} Q$ *in* $\overline{\text{PS}}$. *Consider some partition* PS_q *in which there are either r-productions or a-productions relevant to predicate* Q, *and such that there are no production rules relevant to* Q *in* $\bigcup_{i>q} \text{PS}_i$. *Then predicate* P *must be distinguished in* $\bigcup_{j \le q} \text{PS}_j$.

PROOF. The proof of this lemma trivially follows from the fact that the sequence of rules in $\overline{\text{PS}}$ such that either of these conditions $P \rightarrow^{+} Q$ or $P \rightarrow^{-} Q$ holds, must be derived from production rules in PS. Further, P must occur in the body of at least one of these production rules. Finally, these production rules must all be contained within $\bigcup_{j \le q} \text{PS}_j$, else the conditions for PS to be stratified will be violated. Thus, P must be distinguished within $\bigcup_{j \le q} \text{PS}_j$. □

Theorem. If PS *is a stratified program, then the corresponding logic program* $\overline{\text{PS}}$, *obtained from* PS *using the transformations* TR_{init} *and* TR_{cons}, *is stratified.*

PROOF. Recall that using the previously described transformations TR_{init} and TR_{cons}, r-productions of the form

$$(p_1 \quad P, C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow \textbf{retract } P)$$

[2] These definitions are in the spirit of dependency graphs for logic programs which are defined in [1].

and a-productions of the form

$$(p_2 \quad A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b \rightarrow \textbf{assert } P)$$

derive clauses of the following form in $\overline{\text{PS}}$ (where we assume that the two literals in the heads of the rules are unifiable):

$$C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow P^*,$$

$$A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b, \neg P^* \rightarrow P,$$

where P^* is a previously unused predicate associated with each P.

Now, for $\overline{\text{PS}}$ to be a stratified program, the condition $P \rightarrow^- P$ must not hold. Thus, the following conditions must be satisfied in the logic program $\overline{\text{PS}}$:

1. It must not be the case that $P \rightarrow^+ X$ or $P \rightarrow^- X$ in $\overline{\text{PS}}$, where X is some C_k or some D_k occurring in some rule in $\overline{\text{PS}}$ defining P^*.
2. It must not be the case that $P \rightarrow^- X$, where X is some literal A_k (occurring positively) in some rule in $\overline{\text{PS}}$ defining P.
3. It must not be the case that $P \rightarrow^+ X$ or $P \rightarrow^- X$, where X is some literal B_k (occurring negatively) in some rule in $\overline{\text{PS}}$ defining P.

We will see that each of these conditions is indeed satisfied.

Let the r-productions such as p_1 relevant to P occur in some partition PS_x. Now, all production rules that are relevant to any C_k or D_k must occur in $\bigcup_{p \leq x} \text{PS}_p$ to satisfy the condition for the PS program to be stratified. Now, suppose that condition 1 is not satisfied in $\overline{\text{PS}}$, i.e., $P \rightarrow^+ X$ or $P \rightarrow^- X$, where X is some C_k or D_k in p_1. From Lemma 1 we have that predicate P is then distinguished in some partition $\bigcup_{q \leq p} \text{PS}_q$. However, there is already an r-production in PS_x relevant to P, and we have $q \leq x$. This violates both the conditions 3 and 5 for the program PS to be stratified. It follows that condition 1 must hold.

Next, consider the a-productions such as p_2 relevant to P. They must occur in some partitions $\bigcup_{y \leq x} \text{PS}_y$ if PS is to be stratified. In addition, all production rules relevant to either A_k or B_k must occur in $\bigcup_{r \leq y} \text{PS}_r$ if PS is to be stratified. Again, if either condition 2 or 3 is not satisfied $\overline{\text{PS}}$, then it follows from Lemma 1 that the predicate P is distinguished in $\bigcup_{s \leq r} \text{PS}_s$. Since $s \leq r$, $r \leq y$, and $y \leq x$, it follows that P is distinguished in $\bigcup_{s \leq x} \text{PS}_s$. This too violates conditions 3 and 5 for program PS to be stratified, since there is already an r-production relevant to P in PS_x. It follows that conditions 2 and 3 must hold. □

Lemma 2. If $\text{PS} = \text{PS}_1 \cup \cdots \cup \text{PS}_n$ *is stratified, then* $\overline{\text{PS} - (i)}$, *the logic program corresponding to production rules in* $\text{PS}_1 \cup \cdots \cup \text{PS}_i$ *is stratified, for* $1 \leq i \leq n$.

PROOF. The proof of this lemma trivially follows from the criterion for stratification. □

Definition. $M_{\overline{\text{PS}}}$ is the standard minimal model for the stratified logic program $\overline{\text{PS}}$.

Lemma 3. Let $\overline{\text{PS} - 1}$ *and* $\overline{\text{PS} - 2}$ *be two stratified logic programs such that* $\overline{\text{PS} - 1} \subseteq \overline{\text{PS} - 2}$. *Then* $M_{\overline{\text{PS}-1}} \subseteq M_{\overline{\text{PS}-2}}$.

PROOF. The proof of Lemma 3 follows from the definition of the standard minimal model for a stratified logic program [1].

Definition. Let $S_{\text{pos}}(i,j) = \{X$: **assert** $X \in T_{\text{PS}_j} \uparrow i$ (EDB_{j-1}) and X is *distinguished* and *not retractable* in $\text{PS}_j\}$.

To explain, $S_{\text{pos}}(i,j)$ are the literals that are distinguished in partition PS_j and are entailed after the operator T_{PS} has been applied i times to the production rules in PS_j. Further, there are no r-productions relevant to X in PS_j.

Definition. $S_{\text{neg}}(i,j) = \{X$: **assert** $X \notin T_{\text{PS}_j} \uparrow i$ (EDB_{j-1}) and X is *distinguished* and *not retractable* in $\text{PS}_j\}$.

Lemma 4. Each application of the T_{PS} operator with the production rules in PS_j will either cause the set $S_{\text{pos}}(i,j)$ to grow or to stay the same, as follows:

$S_{\text{pos}}(i,j) \subseteq S_{\text{pos}}(i+1,j)$ *for all* $0 \le i \le \omega$,

EDB_j *entails* $\{X\colon X \in S_{\text{pos}}(\omega,j)\}$.

PROOF. To prove this, we observe from the definition that $S_{\text{pos}}(0,j) = \{X$: EDB_{j-1} entails $X\}$. Since each literal X is distinguished and not retractable in PS_j, we observe that there can be no r-production in PS_j relevant to X. It follows that the set S_{pos} cannot shrink and either grows or stays fixed with each application of the operator. Consequently the following holds:

$S_{\text{pos}}(i,j) \subseteq S_{\text{pos}}(i+1,j)$ for all $0 \le i \le \omega$,

EDB_j entails $\{X\colon X \in S_{\text{pos}}(\omega,j)\}$. □

Lemma 5. Each application of the T_{PS} operator to the production rules in PS_j will cause the set $S_{\text{neg}}(i,j)$ to shrink or to stay the same, as follows:

$S_{\text{neg}}(i+1,j) \subseteq S_{\text{neg}}(i,j)$ *for all* $0 \le i \le \omega$,

EDB_j *does not entail* $\{X\colon X \in S_{\text{neg}}(\omega,j)\}$.

Literals that occur negatively in the body of production rules in PS_j remain in $S_{\text{neg}}(\omega,j)$, the fixpoint for PS_j, if these literals are in $S_{\text{neg}}(0,j)$, as follows:

$\{X$: *X occurs negatively in the body of some production rule in* PS_j *and X is in* $S_{\text{neg}}(0,j)\} \subseteq S_{\text{neg}}(\omega,j)$.

PROOF. To prove this, we observe from the definition that $S_{\text{neg}}(0,j) = \{X$: EDB_{j-1} does not entail $X\}$. Since each literal X is distinguished and not retractable in PS_j, there can be no r-production in PS_j that is relevant to X. Thus, the set S_{neg} cannot grow and must either shrink or stay fixed with each application of the operator. Consequently, the following holds:

$S_{\text{neg}}(i+1,j) \subseteq S_{\text{neg}}(i,j)$ for all $0 \le i \le \omega$,

EDB_j does not entail $\{X\colon X \in S_{\text{neg}}(\omega,j)\}$.

From condition 4 that ensures PS is stratified, we observe that when some distinguished literal X occurs negatively in the body of some production rule in

PS_j, then PS_j cannot contain any a-productions which are relevant to X. From this, the following condition is obtained:

$\{X$: X occurs negatively in the body of some production rule in PS_j and X is in $S_{neg}(0, j)\} \subseteq (S_{neg}(\omega, j))$. □

Lemma 6. A predicate X that is distinguished *in* PS_j *and occurs in the body of a production rule q in* PS_j *that is not relevant to X, is* not retractable *in* PS_j.

PROOF. The proof of this lemma trivially follows from the conditions 3 and 5 for PS to be stratified. □

Definition. A production rule in PS_j is determined *executable* in the ith step, i.e., the ith application of the operator, $T_{PS_j} \uparrow i\ (EDB_{j-1})$, when all literals occurring positively in the body of this production rule are entailed after the $(i-1)$th application of the operator, $T_{PS_j} \uparrow i-1\ (EDB_{j-1})$, and all literals occurring negatively in the body of the production rule are not entailed.

Lemma 7. Let an executable *production rule p in* PS_j *be executed in the ith step, where p could be*

$(p\ \ A_1, \ldots, A_a, \ldots, \neg B_1, \ldots, \neg B_b \rightarrow$ **assert/retract** $P)$.

Let $\overline{PS-(j-1)}$ *be the logic program corresponding to the production rules in the previous* $(j-1)$ *partitions* $PS_1 \cup \cdots \cup PS_{j-1}$ *with the standard model* $M_{\overline{PS-(j-1)}}$.

Assume that EDB_{j-1} *entails* $M_{\overline{PS-(j-1)}}$, *i.e., it entails Q where Q is true in* $M_{\overline{PS-(j-1)}}$ *and does not entail Q where Q is false in* $M_{\overline{PS-(j-1)}}$, *ignoring the special predicates that are introduced into the logic program* $\overline{PS-(j-1)}$. *Let* $\overline{PS-(j)}$ *be the program corresponding to the production rules in the j partitions* $PS_1 \cup \cdots \cup PS_j$ *with the standard model* $M_{\overline{PS-(j)}}$. *Then all* A_k *must be true in* $M_{\overline{PS-(j)}}$ *and* EDB_j *must entail all* A_k. *Further, all* B_k *must be false in* $M_{\overline{PS-(j)}}$ *and* EDB_j *must not entail any* B_k.

PROOF

Base Case. Let production rule p be executed in step 1. This implies that EDB_{j-1} entails each A_k and does not entail each B_k. Since we assume EDB_{j-1} entails $M_{\overline{PS-(j-1)}}$, it follows that $\overline{PS-(j-1)}$ must prove each A_k. Applying Lemma 6, we observe that there are no r-productions relevant to A_k, which can transform either the fact A_k or corresponding rules that define A_k, in $\overline{PS-(j-1)}$. Applying Lemma 3, each A_k must be true in $M_{\overline{PS-(j)}}$. A similar argument holds for B_k. Since EDB_{j-1} does not entail each B_k and since we assume EDB_{j-1} entails $M_{\overline{PS-(j-1)}}$, it follows that $\overline{PS-(j-1)}$ cannot prove each B_k. By condition 4 for PS to be stratified, there can be no a-productions relevant to B_k in PS_j, and so there are no new rules in $\overline{PS-(j)}$ defining each B_k. Applying Lemma 3, each B_k remains false in $M_{\overline{PS-(j)}}$.

By the definition that p executes in step 1, each A_k is in $S_{pos}(0, j)$ and each B_k is in $S_{neg}(0, j)$. Applying Lemmas 4 and 5, we have that EDB_j entails each A_k and does not entail each B_k.

Inductive Case. For the inductive case, suppose p is executed in some step i. From the definition of the inductive case, each A_k is either entailed by EDB_{j-1} or is entailed by some production rule p_{i-1} which adds (**assert** A_k) in some step $i-1$ or some previous step. For those A_k entailed by EDB_{j-1}, we can use the argument

previously used in the base case to show that A_k is true in $M_{\overline{PS-(j)}}$ and that EDB_j entails A_k.

Suppose that some A_k is entailed by some production rule p_{i-1}, in the previous step $i-1$. From the definition of the inductive case, each C_k occurring positively in the body of such a p_{i-1} will be true in $M_{\overline{PS-(j)}}$ and each D_k occurring negatively in the body of such a p_{i-1} will be false. Applying Lemma 6, there can be no r-productions relevant to A_k in PS_j. Each rule in $\overline{PS-(j)}$ defining A_k, derived from each a-production p_{i-1} in PS_j, will not be transformed by TR_{cons}. These rules defining A_k remain unchanged in $\overline{PS-(j)}$. Applying Lemma 3, each A_k will be true in $M_{\overline{PS-(j)}}$.

By the definition that p is executed in step i, we have that each A_k must be in $S_{pos}(i-1, j)$. Applying Lemma 4, we have that EDB_1 entails each A_k.

Further, since p is executed in step i, we have by definition that each B_k is in $S_{neg}(i-1, j)$. Applying Lemma 5, it follows that each B_k is also in $S_{neg}(0, j)$, i.e., EDB_{j-1} does not entail each B_k. Since we assume that EDB_{j-1} entails $M_{\overline{PS-(j-1)}}$, we know that $\overline{PS-(j-1)}$ cannot prove each B_k. From condition 4 for PS to be stratified, there can be no a-productions relevant to each B_k in PS_j. Thus, there can be no new rules in $\overline{PS-(j)}$ defining each B_k. Applying Lemma 3, each B_k will reach remain false in $M_{\overline{PS-(j)}}$. Further, each B_k is in $S_{neg}(0, j)$. By Lemma 5, EDB_j will not entail each B_k. □

Definition. For two production rules p and q, in some partition PS_j, we say $\text{prec}(p) \leq \text{prec}(q)$ if p is relevant to some literal A_k occurring in the body of q, or, for some production rule r in PS_j, $\text{prec}(p) \leq \text{prec}(r)$ and $\text{prec}(r) \leq \text{prec}(q)$.

Definition. For two production rules p and q, $\text{prec}(p) = \text{prec}(q)$ if $\text{prec}(p) \leq \text{prec}(q)$ and $\text{prec}(q) \leq \text{prec}(p)$. Further, $\text{prec}(p) < \text{prec}(q)$ if $\text{prec}(p) \leq \text{prec}(q)$ and $\text{prec}(p) \neq \text{prec}(q)$.

To motivate the next two definitions, it is necessary to determine when a production rule p in some partition PS_j cannot be executed in some step i, i.e., the ith application of the T_{PS} operator in PS_j, and when this production rule p can never be executed in any subsequent step k, $i < k < \omega$, until the fixpoint EDB_j is obtained. To do so, we present definitions for when a production rule is determined not executable, denoted *n-e* and when a production rule is determined to remain not executable, denoted *r-n-e*, as follows:

Definition. A production rule p in PS_j relevant to the predicate P is determined *not executable*, denoted *n-e*, in the ith step if at least one literal A_k occurring positively in the body of p is not entailed after the $(i-1)$th application of the operator, $T_{PS_j} \uparrow i-1\ (EDB_{j-1})$ or if at least one literal B_k, occurring negatively, is entailed. Further, all other a-productions in PS_j, say q, relevant to each literal A_k (which is not retractable) and all other production rules relevant to the literal P, where $\text{prec}(q) < \text{prec}(p)$, must satisfy one of the following conditions:

1. q is executed before step i.
2. q is determined to be *r-n-e* before step i (the definition of *r-n-e* follows).
3. The execution of q is irrelevant and has no effect on the execution of p. This occurs when there are several a-productions relevant to some A_k. When one of them is executed, executing the others is irrelevant.

NOTE. From conditions 4 and 5 for PS to be a stratified program, there can be no production rules relevant to some B_k occurring negatively in the body of p in PS_j. Further, applying Lemma 6, all literals A_k are not retractable in PS_j and so there can be no r-productions relevant to A_k. Thus, we ignore the effect of such production rules.

Definition. A production rule p *remains not executable*, denoted *r-n-e*, in step i and subsequent steps, if p is determined to be n-e in step i, **and** if all other a-productions, say q in PS_j, that are relevant to some A_k in the body of p, which is not retractable, and where $\text{prec}(q) = \text{prec}(p)$, satisfy one of the following conditions: (1) q is executed before step i or (2) q is determined to be n-e in step i or (3) the execution of q is irrelevant.

NOTE. Suppose we construct a chain of a-productions $p_1, p_2, \ldots, p_n$ in partition PS_j such that each a-production p_i in the chain is relevant to a literal X_i occurring positively in the body of the next a-production in the chain, p_{i+1}. Suppose this chain results in a cycle (or several cycles) of such a-productions, i.e., $p_1 = p_n$. For each pair of production rules, p_i and p_{i+1}, in this cycle, $\text{prec}(p_i) = \text{prec}(p_{i+1})$. Suppose the literals X_i in each production rule (that were used in constructing the cycle) are not entailed by EDB_{j-1} nor are they entailed by the execution of any other a-productions relevant to X_i, say r, where $\text{prec}(r) < \text{prec}(p_i)$. In this case, the execution of each a-production in the cycle is dependent on the execution of all other a-productions in the cycle. All these a-productions in the cycle will simultaneously be determined to be r-n-e in the same step, and none of these X_i will be entailed. The rules of the logic program that are derived from the a-productions in the cycle will also give the condition $X_i \rightarrow^+ X_i$, for each X_i. We also note that all literals X_i which are used to construct the cycle must be not retractable in PS_j. If this is not so, then the conditions that PS is stratified will be violated.

Lemma 8. Let a production rule p in PS_j *be determined r-n-e in the ith step, where p is*

$$(p \quad A_1, \ldots, A_a, \ldots, \neg B_1, \ldots, \neg B_b \rightarrow \textbf{assert/retract } P).$$

Let $\overline{PS - (j-1)}$ *be the logic program corresponding to the production rules in the previous partitions* $PS_1 \cup \cdots \cup PS_{j-1}$ *with the standard model* $M_{\overline{PS-(j-1)}}$. *Assume that* EDB_{j-1} *entails* $M_{\overline{PS-(j-1)}}$. *Let* $\overline{PS - (j)}$ *be the program corresponding to the production rules in the j partitions* $PS_1 \cup \cdots \cup PS_j$ *with the standard model* $M_{\overline{PS-(j)}}$. *Then at least one* A_k *must be false in* $M_{\overline{PS-(j)}}$ *and* EDB_j *must not entail this* A_k, *or at least one* B_k *must be true in* $M_{\overline{PS-(j)}}$ *and* EDB_j *must entail this* B_k.

The proof of Lemma 8 is in the Appendix.

Lemma 9. If P is distinguished in some partition PS_j, *then the following hold*:

- *If* EDB_j *entails P, then* EDB_k *entails P,* $j \leq k \leq n$.
- *If* EDB_j *does not entail P, then* EDB_k *does not entail P,* $j \leq k \leq n$.
- *If P is true in* $M_{\overline{PS-(j)}}$, *then P is true in* $M_{\overline{PS-(k)}}$, $j \leq k \leq n$.
- *If P is false in* $M_{\overline{PS-(j)}}$, *then P is false in* $M_{\overline{PS-(k)}}$, $j \leq k \leq n$.

PROOF. The proof of this lemma trivially follows from the fact that the stratification conditions ensure that there can be neither a-productions nor r-productions relevant to P in $\mathrm{PS}_{j+1} \cup \cdots \cup \mathrm{PS}_n$, when P is distinguished in PS_j. We then observe that there can be no new rules defining P in $\overline{\mathrm{PS} - (k)}$, $j \leq k \leq n$. Also, the rules defining P in $\overline{\mathrm{PS} - (j)}$ [or the fact P in $\overline{\mathrm{PS} - (j)}$] cannot be transformed by $\mathrm{TR}_{\mathrm{cons}}$ and they will remain unchanged in $\overline{\mathrm{PS} - (k)}$, $j \leq k \leq n$. Applying the results of Lemmas 3, 4, and 5, the proof of Lemma 9 follows. □

Theorem. The fixpoint for production rules in PS, EDB_n, *entails the standard minimal model* $M_{\overline{\mathrm{PS}}}$ *for* $\overline{\mathrm{PS}}$, *ignoring the special predicates in* $\overline{\mathrm{PS}}$.

PROOF. To elaborate, when EDB_n entails P, then P must be true in $M_{\overline{\mathrm{PS}}}$. Conversely, when EDB_n does not entail P, then P must be false in $M_{\overline{\mathrm{PS}}}$.

Base Case. The base case is to prove that EDB_0 entails $M_{\overline{\mathrm{PS} - (0)}}$. By definition, there are no production rules in PS_0 and $\overline{\mathrm{PS} - (0)}$ is identical to the initial database $\mathrm{EDB}_{\mathrm{init}}$. The proof trivially follows from the definition of EDB_0.

Inductive Case. Let $\overline{\mathrm{PS} - (i)}$ be the stratified program derived from the production rules in $\mathrm{PS}_1 \cup \cdots \cup \mathrm{PS}_i$. Assume EDB_i entails $M_{\overline{\mathrm{PS} - (i)}}$, ignoring the special predicates in $\overline{\mathrm{PS} - (i)}$. Then we must prove that EDB_{i+1} must entail $M_{\overline{\mathrm{PS} - (i+1)}}$, again ignoring the special predicates. Note that the previous assumption allows us to apply the results of Lemma 7 and 8. There are two subcases, as follows:

Inductive Subcase 1. Suppose EDB_i entails P, i.e., (**assert** P) $\in \mathrm{EDB}_i$ and (**retract** P) $\notin \mathrm{EDB}_i$. Since we assume EDB_i entails $M_{\overline{\mathrm{PS} - (i)}}$ and since EDB_i entails P, P must be true in $M_{\overline{\mathrm{PS} - (i)}}$. Now $\overline{\mathrm{PS} - (i)}$ may include the following rules defining P:

$$A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b \rightarrow P.$$

It follows that either $a = b = 0$, i.e., P is a fact, or each of the A_k must be true in $M_{\overline{\mathrm{PS} - (i)}}$ and each of the B_k must be false in $M_{\overline{\mathrm{PS} - (i)}}$, so that P is true in $\overline{\mathrm{PS} - (i)}$. From the assumption that EDB_i entails $M_{\overline{\mathrm{PS} - (i)}}$, it also follows that EDB_i entails each A_k and does not entail any B_k.

Now, PS_{i+1} may contain the following r-productions (the a-productions relevant to P may be overlooked since EDB_i entails P):

$$(p_r \quad P, C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow \textbf{retract } P).$$

The corresponding logic program $\overline{\mathrm{PS} - (i+1)}$ will now include the following rules defining P and P^*, assuming that the literals in the heads of the two rules unify:

$$C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow P^*,$$

$$A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b, \neg P^* \rightarrow P.$$

We observe that each A_k and B_k is distinguished in some PS_j, where $j \leq i$. Applying Lemma 9, each of the A_k must remain true in $M_{\overline{\mathrm{PS} - (i+1)}}$ and EDB_{i+1} must entail each A_k. Similarly, each of the B_k must remain false in $M_{\overline{\mathrm{PS} - (i+1)}}$ and EDB_{i+1} must not entail each B_k. Thus, we can ignore these literals.

Suppose some production rule p_r executes and (**retract** P) is added to EDB_{i+1}. We can overlook the case of several p_r executing simultaneously since they will all add (**retract** P). Now (**assert** P) $\in \mathrm{EDB}_{i+1}$ and (**retract** P) $\in \mathrm{EDB}_{i+1}$. Thus, EDB_{i+1} will not entail P. In addition, applying Lemma 7, we have EDB_{i+1} will entail all C_k corresponding to some p_r and will not entail each D_k for this p_r.

Applying Lemma 7, we also have all C_k for some p_r will be true in $M_{\overline{PS-(i+1)}}$ and all D_k for this p_r will be false in $M_{\overline{PS-(i+1)}}$. It follows that P^* will be true and P will be false in $M_{\overline{PS-(i+1)}}$. Thus, EDB_{i+1} entails $M_{\overline{PS-(i+1)}}$, ignoring the special predicate P^*.

Suppose instead that all p_r are determined r-n-e and cannot execute. Now (**assert** P) $\in EDB_{i+1}$ and (**retract** P) $\notin EDB_{i+1}$. Thus, EDB_1 will entail P. Applying Lemma 8, we have EDB_{i+1} will not entail at least one C_k or it will entail at least one B_k, for each p_r.

Applying Lemma 8, we also have at least one C_k will be false in $M_{\overline{PS-(i+1)}}$ or at least one D_k will be true in $M_{\overline{PS-(i=1)}}$, for each p_r. It follows that P^* will be false and P will be true in $M_{\overline{PS-(i+1)}}$. Consequently, EDB_{i+1} entails $M_{\overline{PS-(i+1)}}$, again ignoring P^*.

Inductive Subcase 2. Suppose EDB_i does not entail P, i.e., either (**assert** P) $\notin EDB_i$ or (**assert** P) $\in EDB_i$ and (**retract** P) $\in EDB_i$. Since we assume that EDB_i entails $M_{\overline{PS-(i)}}$, P must be false in $M_{\overline{PS-(i)}}$.

If we suppose the latter case that (**retract** P) $\in EDB_i$, we observe there is some r-production relevant to P in $\bigcup_{j<i} PS_j$. This implies that P was a distinguished literal in some PS_j, where $j < i$. Applying the results of Lemma 9, it follows that EDB_{i+1} will not entail P and that P is false in $M_{\overline{PS-(i+1)}}$.

Suppose instead that (**assert** P) $\notin EDB_i$. Since P is false in $M_{\overline{PS-(i)}}$, we observe that if $\overline{PS-(i)}$ contained any rules defining P, such as

$$E_1, \ldots, E_e, \neg F_1, \ldots, \neg F_f \rightarrow P,$$

then from the definition of the inductive case, these rules are determined r-n-e. Either some E_k is false in $M_{\overline{PS-(i)}}$ or some F_k is true in $M_{\overline{PS-(i)}}$. Similarly, EDB_i does not entail this E_k or EDB_i entails this F_k. Applying Lemma 9, we observe that either this E_k remains false in $M_{\overline{PS-(i+1)}}$ and EDB_{i+1} does not entail this E_k, or this F_k remains true in $M_{\overline{PS-(i+1)}}$ and EDB_{i+1} entails this F_k. In either case, this rule defining P cannot prove P in $M_{\overline{PS-(i+1)}}$ and can be overlooked.

We now consider any a-productions or r-productions relevant to P in PS_{i+1}, which are

$$(p_a \quad A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b \rightarrow \textbf{assert } P).$$

$$(p_r \quad P, C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow \textbf{retract } P).$$

The corresponding rules in $\overline{PS-(i+1)}$ are

$$C_1, \ldots, C_c, \neg D_1, \ldots, \neg D_d \rightarrow P^*,$$

$$A_1, \ldots, A_a, \neg B_1, \ldots, \neg B_b, \neg P^* \rightarrow P.$$

Suppose at least one p_a executes, but all p_r are determined r-n-e and are not executed. Now, (**assert** P) $\in EDB_{i+1}$ and (**retract**) $\notin EDB_{i+1}$; EDB_{i+1} will entail P. Note that once one production rule such as p_a executes, the other relevant a-productions can be overlooked. Applying Lemmas 7 and 8, we have EDB_{i+1} will entail each A_k and will not entail each B_k, for some p_a. In addition, for each p_r, EDB_{i+1} will not entail at least one C_k or it will entail at least one D_k.

Applying Lemmas 7 and 8, we have each A_k will be true in $M_{\overline{PS-(i+1)}}$ and each B_k will be false for this p_a. In addition, for each p_r, at least one C_k will be false or at least one D_k will be true. Consequently, P^* will be false and P will be true, in

$M_{\overline{PS-(i+1)}}$. Thus, EDB_{i+1} entails $M_{\overline{PS-(i+1)}}$, ignoring the special predicates in $\overline{PS}$. Suppose at least one p_a and at least one p_r execute. As stated earlier, we can overlook the execution of other production rules such as p_a or p_r. Now, (**assert** P) $\in EDB_{i+1}$ and (**retract**) $\in EDB_{i+1}$; EDB_{i+1} will not entail P.

Applying Lemmas 7 and 8, we have EDB_{i+1} will entail each A_k and will not entail each B_k, for some p_a. In addition, for some p_r, EDB_i will entail each C_k and it will not entail each D_k.

Further applying Lemmas 7 and 8, we have each A_k will be true in $M_{\overline{PS-(i+1)}}$ and each B_k will be false for this p_a. In addition, each C_k will be true and each D_k will be true for this p_r. Consequently, P^* will be true and P will be false in $M_{\overline{PS-(i+1)}}$. Thus, EDB_{i+1} entails $M_{\overline{PS-(i+1)}}$, again ignoring the special predicates in $\overline{PS}$.

The case where all a-productiosn and r-productions are determined r-n-e and do not execute is very similar and can be easily analyzed. It cannot be the case that all a-productions such as p_a are determined r-n-e and do not execute but some p_r executes. This is because EDB_i does not entail P, but P occurs positively in the body of each p_r.

Thus, for the two different subcases (EDB_i does not entail P or EDB_i entails P), we have shown that when EDB_i entails $M_{\overline{PS-(i)}}$, then EDB_{i+1} entails $M_{\overline{PS-(i+1)}}$, ignoring the special predicates in $\overline{PS}$. □

6. CONCLUSIONS AND FUTURE RESEARCH

A fixpoint and declarative semantics has been defined for a class of function-free stratified PS programs. Processing of the production rules in the PS program terminates upon reaching a fixpoint EDB_n, based on a defined fixpoint operator T_{PS}. Each stratified PS program is associated with a stratified logic program $\overline{PS}$. The standard minimal model for $\overline{PS}$ is the declarative meaning of the PS program.

We define two transformations TR_{init} and TR_{cons} to obtain $\overline{PS}$ from PS. The standard model for the initial logic program PS_{init} that is derived from the initial database and the a-productions of the PS program (using TR_{init}) may be inconsistent with the set of integrity constraints (IC) that are derived from the r-productions of the PS program (also using TR_{init}). We transform PS_{init} to obtain $\overline{PS}$ using TR_{cons}. $\overline{PS}$ has a standard model $M_{\overline{PS}}$ which is consistent with IC.

$\overline{PS}$ represents the meaning of the PS program. We prove that if PS is stratified, then $\overline{PS}$ is a stratified logic program. The declarative meaning of the PS program is the standard minimal model $M_{\overline{PS}}$ for $\overline{PS}$. We prove the equivalence of the fixpoint and declarative semantics by showing that EDB_n, the fixpoint for the production rules in the stratified PS program, entails $M_{\overline{PS}}$.

In related research, we extend the semantics of PS programs to include *evaluable* functions in the body of production rules, and **modify** actions (corresponding to a database update) in the head of production rules that correspond to integrity constraints [25, 26]. We also consider *sequences* of actions in the head of production rules. These production rules may derive both rules and integrity constraints. These extensions ar discussed in [24]. We also study PS programs that exhibit nondeterministic behavior when executing production rules. The logic programs derived from these programs may not be stratified and there may not be a deterministic (single) answer for these programs. These results are reported in [25,

26]. Finally, we consider techniques for implementing the semantics of production rule programs in a DBMS environment in [22, 24].

APPENDIX A
PROOF OF LEMMA 8

Lemma 8. Let a production rule p in PS_j *be determined r-n-e in the ith step, where p is*

$(p \quad A_1, \ldots, A_a, \ldots, \neg B_1, \ldots, \neg B_b \rightarrow$ **assert/retract** $P)$.

Let $\overline{PS-(j-1)}$ *be the logic program corresponding to the production rules in the previous partitions* $PS_1 \cup \cdots \cup PS_{j-1}$ *with a standard model* $M_{\overline{PS-(j-1)}}$. *Assume that* EDB_{j-1} *entails* $M_{\overline{PS-(j-1)}}$. *Let* $\overline{PS-(j)}$ *be the program corresponding to the production rules in the j partitions* $PS_1 \cup \cdots \cup PS_j$ *with a standard model* $M_{\overline{PS-(j)}}$. *Then at least one* A_k *must be false in* $M_{\overline{PS-(j)}}$ *and* EDB_j *must not entail this* A_k, *or at least one* B_k *must be true in* $M_{\overline{PS-(j)}}$ *and* EDB_j *must entail this* B_k.

PROOF.

Base Case. Let production rule p be determined r-n-e in step 1. This implies that EDB_{j-1} does not entail at last one A_k or it entails at least one B_k. Suppose that p is r-n-e because some A_k is not entailed by EDB_{j-1}. Then this A_k must be false in $M_{\overline{PS-(j-1)}}$. By definition of this base case that p is r-n-e in step 1, either there can be no relevant a-productions for this A_k, or all relevant a-productions q are such that $\text{prec}(p) = \text{prec}(q)$ and each q is determined n-e in step 1.

Suppose that p is r-n-e in step 1 and there are no a-productions relevant to A_k. It follows that there can be no new rules in $\overline{PS-(j)}$ defining A_k. Applying Lemma 6, A_k is not retractable in PS_j. Hence, there can be no r-productions relevant to A_k. Any rules in $\overline{PS-(j-1)}$ defining A_k remain unchanged in $\overline{PS-(j)}$. Applying Lemma 3, this A_k remains false in $M_{\overline{PS-(j)}}$.

Suppose instead that p is n-e in step 1 and all a-productions q relevant to this A_k, where $\text{prec}(p) = \text{prec}(q)$, are also n-e in step 1. Since EDB_{j-1} entails $M_{\overline{PS-(j-1)}}$, it follows that no rule in $\overline{PS-(j-1)}$ can prove this A_k. Applying Lemma 6, we can show that these rules are not changed by TR_{cons} and cannot prove this A_k. In addition, p and each of the a-productions q, where $\text{prec}(p) = \text{prec}(q)$, will have some nonretractable literal X_i occurring positively in its body. By definition that p is n-e in step 1, these X_i will be false in $M_{\overline{PS-(j-1)}}$.

We can construct a cycle(s) of a-productions which includes p and each q. Each a-production in the cycle has some literal X_i which is not entailed by EDB_{j-1}. In addition, the literals X_i can only be entailed by the execution of some other a-production in the cycle(s). Thus, the execution of each a-production is dependent on the simultaneous execution of all other a-productions in the cycle(s). Consequently, it will be the case that all these a-productions in the cycle(s) are determined r-n-e, simultaneously, in step 1. Each X_i, including this A_k, will be in $S_{neg}(1, j)$, and will remain in $S_{neg}(\omega, j)$. Thus, EDB_j will not entail each X_j or A_k.

The rules derived from these a-productions p, and each q, will be such that the following condition holds for each X_j: $X_i \rightarrow^+ X_i$. As a result, these rules will not prove any X_i. These rules will also be unchanged by TR_{cons}, since each X_i is not retractable in PS_{j-1}. Applying Lemma 3, each X_i, including A_k, will remain false in $M_{\overline{PS-(j)}}$.

Suppose that p is r-n-e in step 1 because some B_k is entailed by EDB_{j-1}. B_k is in $S_{pos}(0, j)$. By Lemma 4, EDB_j entails B_k. Since we assume EDB_{j-1} entails

$M_{\overline{PS-(j-1)}}$, it follows that $\overline{PS-(j-1)}$ proves this B_k. Applying Lemma 6, this B_k is not retractable in PS_j. Thus, there are no r-productions relevant to B_k in PS_j. The rules defining B_k in $\overline{PS-(j-1)}$ cannot be transformed by TR_{cons} and remain unchanged in $\overline{PS-(j)}$. Applying Lemma 3, this B_k is true in $M_{\overline{PS-(j)}}$.

Inductive Case. For the inductive case, suppose p is determined r-n-e in some step i. This implies that $T_{PS_j} \uparrow i-1\,(EDB_{j-1})$ does not entail at least one A_k or it entails at least one B_k.

Suppose p is r-n-e because some B_k was entailed. Then, B_k is in $S_{pos}(i-1, j)$ and applying Lemma 4, EDB_j will entail B_k. We can use previous arguments as in the base case to prove that this B_k must be true in $M_{\overline{PS-(j-1)}}$ and remains true in $M_{\overline{PS-(j)}}$.

Suppose instead that p was r-n-e in step i because some A_k was not entailed. This implies that all relevant production rules, say p_{i-1}, where $\text{prec}(p_{i-1}) < \text{prec}(p)$, must be determined r-n-e in step $(i-1)$ or some previous step, and all relevant a-productions, say q, where $\text{prec}(p) = \text{prec}(q)$, must be determined n-e in step i.

Suppose p is determined to be r-n-e and all p_{i-1} relevant to A_k are r-n-e in step $i-1$. A_k is in $S_{neg}(i-1, j)$ and applying Lemmas 5 and 6, EDB_j will not entail A_k. From the definition of the inductive case, some literal C_k, occurring positively in the body of each p_{i-1}, will be false in $M_{\overline{PS-(j)}}$, and EDB_j will not entail this C_k, or some D_k, occurring negatively in the body of each p_{i-1}, will be true, in $M_{\overline{PS-(j)}}$ and EDB_j will entail this D_k. The rules derived from these production rules will not prove A_k. Applying Lemma 5, A_k is not retractable in PS_j. It follows there will be no r-productions that transform these rules in TR_{cons} and they are unchanged in $\overline{PS-(j)}$. Applying Lemma 3, these rules in $\overline{PS-(j)}$ will not prove this A_k and A_k will be false in $M_{\overline{PS-(j)}}$.

Suppose instead that p is r-n-e in step i and all a-productions relevant to A_k, say q, where $\text{prec}(p) = \text{prec}(q)$ are determined n-e in step i. We can use previous arguments, as used in the base case, to show that each q will have some distinguished literal X_i which is not entailed by $T_{PS_j} \uparrow i-1(EDB_{j-1})$. By the definition that each q is n-e in step i, we can show that any a-productions, say r, relevant to X_i, where $\text{prec}(r) < \text{prec}(p)$, will be determined r-n-e in step $i-1$ or some previous step. Each X_i, including A_k, will be in $S_{neg}(i, j)$ and will remain in $S_{neg}(\omega, j)$. EDB_j will not entail any X_i or A_k. These production rules r derive rules which will not be able to prove X_i in $\overline{PS-(j)}$. In addition, the rules derived from p and all q, where $\text{prec}(p) = \text{prec}(q)$, will be such that the following condition holds: $X_i \rightarrow^+ X_i$. These rules will also not prove X_i in $\overline{PS-(j)}$. Each X_i is not retractable in PS_j, and the rules defining each X_i in $\overline{PS-(j)}$ will not be transformed by TR_{cons}. Applying Lemma 3, it follows that each X_i, which includes A_k, will be false in $M_{\overline{PS-(j)}}$. □

I would like to thank Anne Litcher, Jorge Lobo, Raymond Ng, and Timos Sellis for many fruitful discussions.

REFERENCES

1. Apt. K. R., Blair, H. A., and Walker, A., Towards a theory of declarative knowledge, in: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, Los Altos, CA, 1988.

2. Bocca, J., EDUCE-A marriage of convenience: Prolog and a relational DBMS, in: *Proceedings of the Third IEEE Symposium on Logic Programming*, IEEE, New York, 1986.
3. Bocca, J., On the evaluation strategy of EDUCE, in: *Proceedings of the ACM Sigmod International Conference on the Management of Data*, 1986.
4. Campbell, S. D. and Olson, S. H., WX1: An expert system for weather radar interpretation, in: *Coupling Symbolic and Numerical Computing in Expert Systems*, 1986.
5. Demolombe, R., Syntactical Characterization of a Subset of Domain Independent Formulas, Technical Report, ONERA-CERT, Toulouse, 1982.
6. Delcambre, L. M. L. and Etheredge, J. N., A self-controlling interpreter for the relational production language, in: *Proceedings of the ACM Sigmod Internationa Conference on the Management of Data*, 1988.
7. Forgy, C. L., OPS5 User's Manual, Technical Report CMU-CS-81-135, Carnegie-Mellon University, 1981.
8. Forgy, C. L., Rete: A fast algorithm for the many pattern/many object pattern match problem, *Artificial Intelligence* 19 (1982).
9. Greismer, J. H. et al., YES/MVS: A continuous real time expert system, in: *Proceedings of the AAAI National Conference on Artificial Intelligence*, 1984.
10. Hayes-Roth, F., Rule based systems, *Commun. ACM* 28:9 (1985).
11. Kershberg, L. (ed.), *Expert Database Systems: Proceedings From the First International Workshop*, Benjamin Cummings, Menlo Park, CA, 1986.
12. Kershberg, L. (ed.), *Expert Database Systems: Proceedings From the First International Conference*, Benjamin Cummings, Menlo Park, CA, 1988.
13. Kohli, M., Giuliano, M., and Minker, J., An overview of the PRISM project. *Computer Architecture News* 15:35–42 (1987).
14. Kowalski, R. and Sadri, F., Knowledge Representation without Integrity Constraints, Technical Report, Department of Computing, Imperial College of Science and Technology, 1989.
15. Kowalski, R. and Sadri, F., Logic programs with exceptions, in: *Proceedings of the International Conference on Logic Programming*, 1990.
16. Maindreville, C. de and Simon, E., A prodution rule based approach to deductive databases, in: *Proceedings of the Fourth International Conference on Data Engineering*, 1988.
17. McDermott, J., R1: A rule based configurer of computer systems, *Artificial Intelligence* 19:39–88 (1982).
18. Minker, J. (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufman, Los Altos, CA, 1988.
19. Morris, K., Ullman, J., and van Gelder, A., Design overview of the NAIL system, in *Proceedings of the Third International Conference on Logic programming*, 1986.
20. Nicolas, J. M., Logic for improving integrity checking in relational data bases, *Acta Informatica* 18(3) (1982).
21. Nicolas, J. M., and Demolombe, R., On the Stability of Relational Queries, Technical Report, ONERA-CERT, Toulouse, 1983.
22. Pang, P. and Raschid, L., Magic implementations of stratified update rule programs, 1992, unpublished.
23. Raschid, L., Maintaining consistency in a stratified production system program, in: *Proceedings of the AAAI National Conference on Artificial Intelligence*, 1990.
24. Raschid, L., Correct implementation of update rule semantics in a database environment, 1992, unpublished.
25. Raschid, L. and Lobo, J., A semantics for a class of non-deterministic and causal production system program, *Automated Reasoning* May. To appear in the Journal of Automated Reasoning.
26. Raschid, L. and Lobo, J., Fixpoint and stable model semantics for update rule programs and implementation in a relational database management system. To appear in the ACM Transactions on Database Systems.

27. Raschid, L., Sellis, T., and Lin, C-C., Exploiting concurrency in a DBMS implementation for production systems, in: *Proceedings of the International Symposium on Databases in Parallel and Distributed Systems*, Austin, TX, 1988.
28. Sellis, T., Lin, C-C., and Raschid, L., Implementing large production systems in a DBMS environment: Concepts and algorithms, in: *Proceedings of the ACM Sigmod International Conference on the Management of Data*, 1988.
29. Simon, E. and Maindreville, C. de, Deciding whether a production rule is relational computable, in *Proceedings of the International Confernece on Database Theory*, Bruges, Belgium, 1988.
30. Tsur, S. and Zaniolo, C., LDL: A logic-based data language, in: *Proceedings of the Twelfth International Conference on Very Large Data Bases*, 1986.
31. van Emden, M. H. and Kowalski, R. A., The semantics of predicate logic as a programming language, *J. ACM* 23(4):733–742 (1976).
32. Widom, J. and Finkelstein, S. J., A syntax and semantics for set-oriented production rules in relational database systems, IBM Research Report, IBM Almaden Research Center, June 1989.